## USN : 1BM22CS259

## LAB-1 : Genetic Algorithm for Optimization Problems

**CODE:**

```
import numpy as np
import random

def objective_function(x):
    return x ** 2

population_size = 100
num_generations = 50
mutation_rate = 0.1
crossover_rate = 0.7
range_min = -10
range_max = 10

# Create initial population
def initialize_population(size, min_val, max_val):
    return np.random.uniform(min_val, max_val, size)

# Evaluate fitness of the population
def evaluate_fitness(population):
    return np.array([objective_function(x) for x in population])

# Selection using roulette-wheel method
def selection(population, fitness):
    total_fitness = np.sum(fitness)
    probabilities = fitness / total_fitness
    return population[np.random.choice(range(len(population)), size=2, p=probabilities)]
```

```
# Crossover between two parents
def crossover(parent1, parent2):
    if random.random() < crossover_rate:
        return (parent1 + parent2) / 2  # Simple averaging for crossover
    return parent1  # No crossover

# Mutation of an individual
def mutate(individual):
    if random.random() < mutation_rate:
        return np.random.uniform(range_min, range_max)
    return individual

# Genetic Algorithm function
def genetic_algorithm():
    # Step 1: Initialize population
    population = initialize_population(population_size, range_min, range_max)

    for generation in range(num_generations):
        # Step 2: Evaluate fitness
        fitness = evaluate_fitness(population)

        # Track the best solution
        best_index = np.argmax(fitness)
        best_solution = population[best_index]
        best_fitness = fitness[best_index]

        # print(f"Generation {generation + 1}: Best Solution = {best_solution}, Fitness =
{best_fitness}")
```

```
        # Step 3: Create new population
        new_population = []
        for _ in range(population_size):
            # Select parents
            parent1, parent2 = selection(population, fitness)
            # Crossover to create offspring
            offspring = crossover(parent1, parent2)
            # Mutate offspring
            offspring = mutate(offspring)
            new_population.append(offspring)

        # Step 6: Replace old population with new population
        population = np.array(new_population)

    return best_solution, best_fitness

# Run the Genetic Algorithm
best_solution, best_fitness = genetic_algorithm()
print(f"Best Solution Found: {best_solution}, Fitness: {best_fitness}")
```

**OUTPUT:**